Threshold Method of Detecting Long-Time TPM Synchronization

Michał Dolecki[1] and Ryszard Kozera[1,2]

[1] The John Paul II Catholic University of Lublin
ul. Konstantynów 1 H; 20-708 Lublin, Poland
michal.dolecki@kul.pl
[2] Warsaw University of Life Sciences – SGGW
ul. Nowoursynowska 159, 02-776 Warsaw, Poland
ryszard.kozera@gmail.com, ryszard_kozera@sggw.pl

Abstract. The phenomenon of neural networks synchronization by mutual learning can be used to construct key exchange protocol on an open channel. For security of this protocol it is important to minimize knowledge about synchronizing networks available to the potential attacker. The method presented herein permits evaluating the level of synchronization before it terminates. Subsequently, this research enables to assess the synchronizations, which are likely to be considered as long-time synchronizations. Once that occurs, it is preferable to launch another synchronization with the new selected weights as there is a high probability (as previously shown) that a new synchronization belongs to the short one.

Keywords: neural networks, Tree Parity Machine, key exchange protocol.

1 Introduction

One of the most important aspects of information security is ensuring the confidentiality of communications. Two parties wish to exchange certain information in such a way, that unauthorized persons have no opportunity to guess the content of the communication. Confidentiality is implemented mainly through data encryption with usage of various cryptographic algorithms [1], [2]. The cryptographic keys are additional input data used in the algorithms to encrypt and decrypt transmitted messages. Due to the keys used, the cryptography can be classified into two types, i.e. symmetric and asymmetric cryptography [1]. Asymmetric cryptography requires from both parties to maintain two keys, one public and the second a private one. Public key is used to encrypt messages, and private one, to decrypt them. Symmetric cryptography uses the same key to both of these operations. In this scheme the security of key distribution and its storage becomes a vital issue. This leads to a paradox: on one hand if one strives to build a secure communication channel, one must have a secure key distribution channel first. On the other hand, if one has secure channel, then why not use it to the communication. To cope with this problem, the key exchange protocol exploiting an open channel can be used.

One of the most popular algorithms used for such key agreement is the Diffie-Hellman algorithm [1], [2], which is based on computationally difficult problem of calculating discrete logarithms in cyclic groups. Such kind of problems cannot be solved in a reasonable amount of time [3]. More specifically there is no proposed algorithm to solve these problems which operates in polynomial time but there may be an algorithm with greater computational complexity. Results presented by three physicists: E. Kanter, W. Kinzel and I. Kanter [4] permit constructing the relatively secure [5-8] key exchange protocol using the phenomenon of artificial neural networks' synchronization by mutual learning.

The idea of applying neural networks in cryptology has emerged relatively recently [9], [10]. This approach represents an interesting alternative to the currently used algorithms based on number theory [11]. In the classical setting, artificial neural networks are built of interconnected layers of neurons. The input network's layer receives input signals and sends them in turn to the first hidden layer's neurons. The output values of neurons of one layer form the inputs for the next neurons' layer etc. Finally, the output of the last layer is the output of the entire network. Each impulse sent to the single neuron is modified by a certain weight. For each neuron, if the sum of weighted impulses exceeds a given threshold level, then the neuron transmits a signal (due to the specifically tuned up activation function). Network classifiers [12-14] receive impulses which belong to one of several selected classes. The main characteristic feature of neural networks is their ability to learn from the presented examples. Such learning process consists of modifying the weights of the neurons to establish the most accurate classification of input impulses. The network is considered as a trained one, if the amount of misclassified input vectors is lower than the prescribed a priori specific acceptance level. For weights, that are real numbers, there are infinitely many different values ensuring a proper output of the given network. The specific and proper use of the network, in practice requires the additional conditions imposed on their structure (i.e. its topology) and neuron activation functions [4-7], [11], [15].

In this paper we present the idea of key exchange protocol using neural networks, where the corresponding weights upon pertinent iterative procedure are set to be equal. We discuss first two methods measuring an overlap of both weight vectors by calculating first the cosine of the angle between them and then alternatively by using the Euclidian metric. Evidently, two approaches mentioned above rely on the knowledge of both networks weights' values which need to be sent via open channel. This, however is impossible in practical key exchange protocol as it would result in security breach. In order to make the weights publicly unknown we analyze the corresponding frequencies of publicly known common outputs of both networks (called here TPMs). Results presented in this paper indicate strong correlation between calculated frequencies and other methods for evaluating TPM's synchronization state. The latter lays foundation to formulate condition for classification of synchronization duration for a given time classes, especially to detect at early stage a long-term synchronization.

2 Tree Parity Machine Synchronization

The mentioned above, proposed by three physicists Kanter, Kinzel and Kanter cryptographic key exchange protocol in an open communication channel which resorts to the so-called Tree Parity Machine network (abbreviated here to TPM). The key exchange procedure abides to the following pattern: two sides of the communication i.e. A and B entities create a special TPM network with the same structure. Both networks start with randomly chosen initial weight vectors w^A and w^B, this initial state is kept secret. Both partners A and B apply a common and publicly available input vector and calculate next the results of their networks. In the next step they exchange their network's results on an open channel. Each party treats the result of the network of another side as the expected result for himself and teaches its own network accordingly (by selecting one of the agreed before learning scheme). The next step is to choose randomly new input vector and the above procedure is subsequently repeated. Upon performing some number of such cycles, both networks' weights coincide and they can be used as cryptographic keys for the established communication. This bidirectional interaction of two synchronizing networks leads to reaching much faster equal weights vectors, than unidirectional one, which potential attacker could also do.

TPM defines a multilayer, feed-forward network with characteristic tree-like structure created by non-overlapping receptive fields. Figure 1 shows the structure of the typical TPM network.

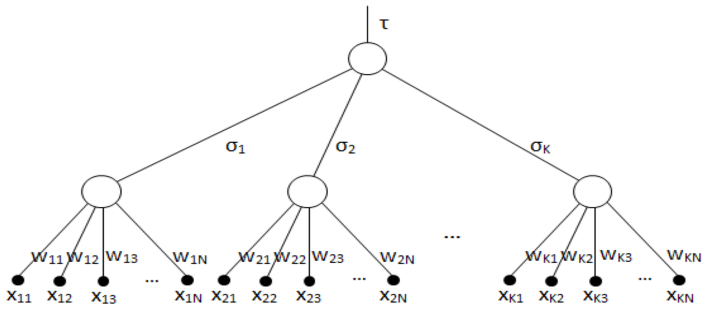

Fig. 1. TPM structure

The hidden layer of this network contains K neurons, each of which has N input signals. The input $x_{ij} \in \{-1,1\}$, where $1 \le i \le K$ and $1 \le j \le N$. Thus, the entire network is made up of the KN inputs. Each input signal is multiplied by the corresponding weight $w_{ij} \in \{-L, -L+1, \ldots, L-1, L\}$. Hidden layer neurons are equipped with the bipolar, step activation function, given by the following formula:

$$\sigma_i = \begin{cases} -1, if \; \sum_{j=1}^{N} x_{ij} w_{ij} \le 0, \\ \;\; 1, if \; \sum_{j=1}^{N} x_{ij} w_{ij} > 0. \end{cases} \quad (1)$$

Commonly in the literature, the TPM network is shortly described with the aid of three parameters: *K*-*N*-*L*. The last layer neuron multiplies hidden layer neurons' outputs and the result of its action is deemed as outcome of the entire network. This operation is given by following formula:

$$\tau = \prod_{i=1}^{K} \sigma_i. \tag{2}$$

TPM network has *KN* weights being integers within the range from -*L* to *L*. Thus, at each synchronization step, it may take one of $(2L+1)^{KN}$ states. For example, the weights of the network with parameters 3-101-3, takes one of $7^{3\cdot 101} \approx 1{,}16 \cdot 10^{256}$ states. The cryptographic key generated using weights would be $3 \cdot 3 \cdot 101 = 909$ bits long.

The network is considered as trained only if its result is equal to the expected one, like in classical Hebbian method [7], [12]. In mutual learning, the expected values are generated by the other network. Hence our two networks are trained only if they both have equals outputs. Each network involved is trained in accordance with one of three methods:

1. Anti-Hebbian learning rule:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - x_{ij}\sigma_i. \tag{3}$$

2. Hebbian learning rule:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + x_{ij}\sigma_i. \tag{4}$$

3. Random-Walk learning rule:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + x_{ij}. \tag{5}$$

If the new value of the weight is greater than *L*, it is replaced by *L*, and analogously, if the value is less than –*L*, it is replaced with –*L*, respectively. In the first learning rule weights are modified once the results of both networks are different, and the process leads to the network synchronization with opposite vectors *w*. The remaining two methods lead to synchronization with the same weight values.

Each pair of neurons form synchronizing TPMs in each learning step perform one of three admissible weight change named in [16] *quiet, attractive* or *repulsive* step. Let *i* denote the index of neuron in both networks *A* and *B*.

1. $\tau^A \neq \tau^B$ or $\tau^A = \tau^B \wedge \sigma_i^A \neq \tau^A \wedge \sigma_i^B \neq \tau^B$: then *A* and *B* do not change the weights' values at all or don't change the weights of *i*-th neuron. In both these cases it is the so-called "quiet step".
2. $\tau^A = \tau^B \wedge \sigma_i^A = \tau^A \wedge \sigma_i^B = \tau^B$: then the weight vectors w_i^A and w_i^B are modified according to one of the listed above learning rules. The modification applied depends on the common input vector x_i and equal neurons output, so that the change is coordinated, because all these values are the same for both networks. Weight vectors change in the same direction, and this case is called "attractive step".

3. $\tau^A = \tau^B \wedge [(\sigma_i^A = \tau^A \ \wedge \ \sigma_i^B \neq \tau^B) \vee (\sigma_i^A \neq \tau^A \ \wedge \ \sigma_i^B = \tau^B)]$: this means, that the outputs of A and B are the same and one of the following events takes place:
   (a) the analyzed neuron of network A has the same output as the result of a whole network and the corresponding neuron in network B generates an output different than whole network B,
   (b) the i-th neuron of network A has different output, than the whole network A, and in network B the corresponding neuron has output equal to whole network B output.

   In both events, only one neuron, which output is equal to the whole network output, modifies its weights, while the weights of the second network's neuron remain unchanged. In general, this procedure results in a reduction of weight vectors compatibility, and this step has been called "repulsive step".

Synchronized TPM networks end up with the same weight vectors and remain synchronized regardless of the time of further learning. For each input vector both networks return the same result, so in each next learning step both network pass through the learning procedure and almost every time networks modify their weights. If K is odd, one changes here the weight of at least one neuron in both networks. For K even there is only one case in which the weights of entire network are not modified. Indeed the latter occurs when all of the hidden layer neurons return -1 and the entire network return 1. For such specific case no neuron will change its weight. In opposite, any other combinations of hidden layer outputs yield at least one neuron changing its weights. Summing-up, synchronized networks typically change the weight during next learning steps which every time yields the same changes in two networks and thus both networks remain synchronized.

The condition enforcing the termination of network's synchronization is to obtain exchange of consistent results of both networks in a sufficiently long period of time. As examined experimentally (see section Results of this paper), in 15000 analyzed synchronizations of the network with the structure 3-101-3 the longest exchange of consistent results by not synchronized network is 147 steps. Thus to be sure that both networks are already synchronized, they should exchange over 147 consistent outputs.

Synchronization is a stochastic process, and the time required to achieve the consistent weights values depends on the initial weights and the input signals generated at each step. The synchronization times of network with a given structure, creates histograms as shown in figure 2 [11]. Such histogram is created after 15000 synchronizations of the network 3-101-3, where the Random Walk learning rule is applied. The number of classes in the histogram is given by the Hunstberger formula $k = 1 + 3{,}32 \cdot \log N$ (see [17]), where here $N = 15000$ and is doubled subsequently for better readability of the chart. Along X-axis the number of learning cycles needed to achieve networks full synchronization is given. On the other hand along Y-axis the probability of finding TPM synchronized in a given number of cycles is specified.

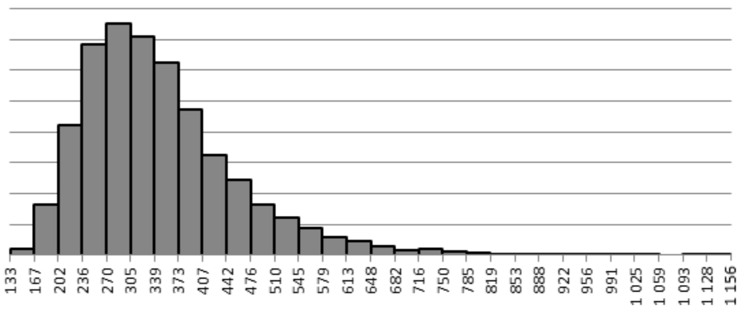

Fig. 2. Times of TPM 3-101-3 synchronization

3 Results

In this paper we analyze the synchronizations of the examined networks with the structure 3-101-3. Such networks enable to agree on key length of 909 bits. The first parameter *K* specifying the number of neurons in a hidden layer is set to 3, which is a typical value used commonly in TPM networks. The parameter *L=3* sets the minimum and maximum weight value *-L* and *L*, respectively. Each weight must therefore be the integer from interval $[-3,3]$, yielding 7 possible values, which is close to the power of 2. This choice of parameter allows to easily generate a fairly uniform distribution of bits with values 0 and 1 in a generated key. The number of bits needed to store the value of the weights is given by $\log_2(2L+1)$ but in fact one has to use $\lceil \log_2(2L+1) \rceil$ bits (here symbol $\lceil\ \rceil$ denote the standard ceiling function). Therefore, it is important to minimize the difference between these values and select *L* which minimizes the following cost function

$$b(L) = \lceil \log_2(2L+1) \rceil - \log_2(2L+1). \tag{6}$$

Figure 3 shows the function *b* depending on the selection of the parameter *L* and taking the values of redundancy in bit representation of weight. It is transparent from the picture that the corresponding value of *L* is given by $2^n - 1$ for some integer *n*.

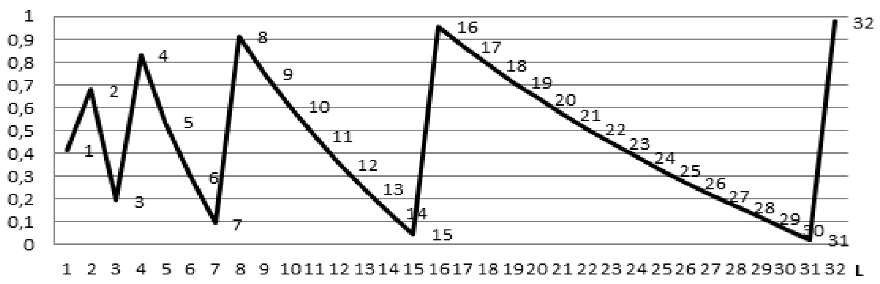

Fig. 3. Function b

Our discussed network is synchronized 15000 times. The fastest synchronization is completed after 133 cycles, while the longest lasted for 1156 cycles. The average synchronization time is in turn 346 cycles. Due to the shape of the histogram (see figure 2) it is interesting to observe that the third quartile value reads as 398, which is slightly above 34% of longest synchronization time. The latter means that 75% of the networks synchronize relatively fast, but there is a group of cases with longer synchronization pattern [18].

Previous research on the synchronization dynamic [7], [16] is based on weights' vectors mutual overlap for each hidden layer neurons. This measurement tool is determined by the cosine of the angle between the vectors of weights (corresponding to the respective neurons in both networks A and B) according to the following formula:

$$\rho_i^{AB} = \frac{w_i^A \circ w_i^B}{\sqrt{w_i^A \circ w_i^A} \cdot \sqrt{w_i^B \circ w_i^B}}. \tag{7}$$

At the initial synchronization phase the cosine renders a value close to 0, while in the end, when TPMs have the same weights, its value is equal to 1. In [19] different approach is presented. Namely, one analyzes the weights' vectors of all neurons and uses the Euclidean distance of these vectors

$$dist(A,B) = \|w^A - w^B\| = \sqrt{\sum_{k=1}^{KN} (w_k^A - w_k^B)^2}. \tag{8}$$

This distance is connected with previously used mutual overlap by cosine theorem for unitary spaces.

This parameter changes during the synchronization from relatively high values for non-synchronized network to 0 for synchronized one. Therefore, it has to be normalized and reversed to preserve the same characteristic as cosine. The normalization operation is accomplished by applying the formula:

$$dist(A,B)_t = 1 - \frac{dist(A,B)_t - \min_{1\le j\le t_{synch}} dist(A,B)_j}{\max_{1\le j\le t_{synch}} dist(A,B)_j - \min_{1\le j\le t_{synch}} dist(A,B)_j}. \tag{9}$$

In figure 4 we present the performance of the Modified Euclidean distance and the cosine measurement for the network 3-101-3 that synchronize within the time equal to the average synchronization time from the all analyzed network's time. The horizontal axis represents TPM's learning time and the vertical one, represents weights compatibility expressed as modified Euclidean distance and cosine.

Reversed normalized distance and cosine have significant linear correlation. Further analysis of this case (see table 3 and 4 below) with the mean square error between the cosine and normalized distance shows that both of these parameters are well-suited to describe the dynamics of synchronization process. In the example mentioned above the mean square error is around 0.016.

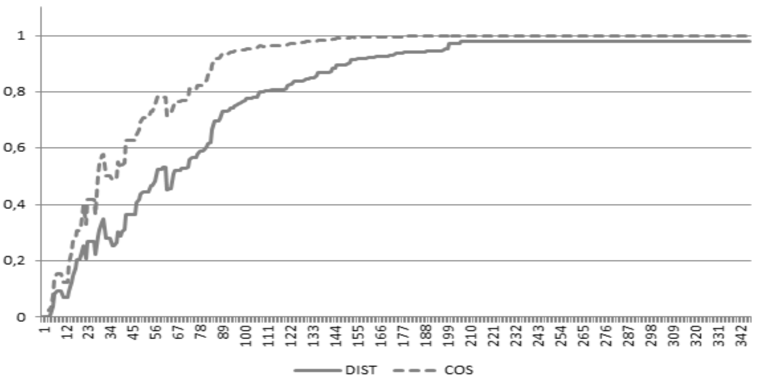

Fig. 4. Modified Euclidean distance and cosine for TPM 3-101-3

The presented above method for modifying distance measurements yield results that stay in good linear correlation with frequency of common output of synchronizing TPMs. Let the (a_n) be a sequence, and $a_n = 1$ if $\tau_n^A = \tau_n^B$ and $\mathrm{a_n} = 0$ if $\tau_n^A \neq \tau_n^B$, $\mathrm{n} = 1,2, \dots, \mathrm{t_{synch}}$. The sequence $(\mathrm{b_n})$ is defined as an average of s elements from $(\mathrm{a_n})$ with indices $\mathrm{n}, \mathrm{n} - 1, \dots, \mathrm{n} - \mathrm{s} + 1$. This sequence is given by formula:

$$b_n = \frac{1}{s}\sum_{j=n-l+1}^{n} a_n, \tag{10}$$

where $l = \min(n, s)$.

During the synchronization process, the weight's vectors of both networks are generally closer, resulting in achieving equal values. The dynamics of these changes can be described by the cosine of the angle between the weights' vectors or by using normalized and reverse Euclidean distance. In both cases, one must know the weights' vectors of the two networks, which is impossible for practical cryptographic key exchange protocol. Linear dependence of the network synchronization level and frequency of the same results exchanged by both networks, permits to treat such frequencies as a tool to determinate how quickly the network will complete the synchronization process. Analyzing these frequencies indicates the threshold at which networks have compatibility level good enough to finish synchronization process in a very short time.

Given 15000 synchronizations of networks with parameters 3-101-3, 5 networks are selected that synchronized after 200 and after 900 cycles, respectively. Visibly, the first group includes the networks that synchronized relatively quickly, while the second one contains networks with the longest observed synchronization time. For each of tested networks we analyzed frequencies of compatible output of the network in 25, 50, 75, ..., 125 previous steps. Next we verified if any of these frequencies exceeds the threshold value 0.7, 0.6, 0.65, ..., 0.85. The tables 1 and 2 illustrate the correlation coefficient between the reversed normalized Euclidean distance and the cosine (see the first column). In addition, they present the correlation coefficient between this distance and frequency calculated for given number of previous steps (the next columns). The green's intensity of the background indicates the best results. Table 1 lists generated herein results for networks synchronized fast in 200 cycles, whereas table 2 presents the same results for networks with synchronization time above 900 cycles. It is transparent once inspecting both tables, that the best fit is obtained for frequencies calculated on the basis of the latest 50-100 steps.

Table 1. Correlation coefficient for TPM synchronized in 200 cycles

	dist, cos	**dist, fr 25**	**dist, fr 50**	**dist, fr 75**	**dist, fr 100**	**dist, fr 125**
TPM 1	0,9675	0,9208	0,9547	0,9726	0,9736	0,9637
TPM 2	0,9463	0,9773	0,9851	0,9809	0,9758	0,9688
TPM 3	0,9452	0,9588	0,9820	0,9790	0,9673	0,9508
TPM 4	0,9543	0,9515	0,9579	0,9652	0,9702	0,9683
TPM 5	0,9583	0,9740	0,9818	0,9722	0,9510	0,9239

Table 2. Correlation coefficient for TPM synchronized in 900 and more cycles

	dist, cos	**dist, fr 25**	**dist, fr 50**	**dist, fr 75**	**dist, fr 100**	**dist, fr 125**
TPM 1	0,9171	0,8560	0,8857	0,8900	0,8875	0,8730
TPM 2	0,9349	0,8804	0,9092	0,9057	0,8998	0,8896
TPM 3	0,9403	0,9084	0,9193	0,9103	0,8887	0,8672
TPM 4	0,9513	0,9215	0,9519	0,9391	0,9227	0,9109
TPM 5	0,9359	0,8479	0,8844	0,8796	0,8544	0,8278

In addition for acquiring high linear relationship between distance and frequency it is also important to determine the compatibility of the above results. In order to achieve the latter we calculated the mean square error for the analyzed network. The corresponding results are demonstrated below in tables 3 and 4. The outcomes from these tables refer to both networks synchronized either in 200 or above 900 cycles, respectively. The obtained output structure is analogous to the previous tables. Again the green's intensity of the background indicates the best results. As previously the best results are obtained for frequencies of 50-100.

Table 3. Mean square error for TPM synchronized in 200 cycles

	dist, cos	**dist, fr 25**	**dist, fr 50**	**dist, fr 75**	**dist, fr 100**	**dist, fr 125**
TPM 1	0,0278	0,0376	0,0188	0,0111	0,0145	0,0256
TPM 2	0,0363	0,0113	0,0047	0,0074	0,0151	0,0273
TPM 3	0,0268	0,0089	0,0079	0,0187	0,0368	0,0598
TPM 4	0,0368	0,0152	0,0114	0,0130	0,0199	0,0319
TPM 5	0,0189	0,0071	0,0057	0,0164	0,0363	0,0624

Table 4. Mean square error for TPM synchronized in 900 and more cycles

	dist, cos	dist, fr 25	dist, fr 50	dist, fr 75	dist, fr 100	dist, fr 125
TPM 1	0,0347	0,0121	0,0095	0,0089	0,0092	0,0108
TPM 2	0,0395	0,0189	0,0148	0,0134	0,0125	0,0124
TPM 3	0,0514	0,0265	0,0232	0,0219	0,0222	0,0229
TPM 4	0,0350	0,0141	0,0098	0,0097	0,0105	0,0112
TPM 5	0,0365	0,0217	0,0169	0,0157	0,0167	0,0180

Figures 5 and 6 illustrate the reverse normalized Euclidean distance calculated from the knowledge of the weights' vectors for both networks and the frequency of reaching the common results by two networks in 50 and 75 previous cycles. There is one network synchronization in 200 and one in over 900 cycles, respectively.

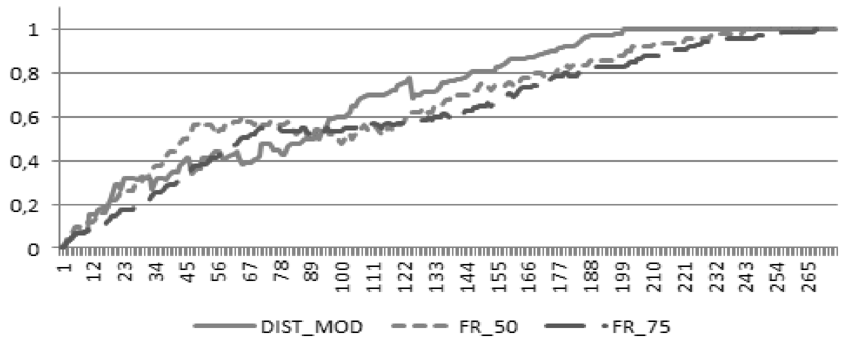

Fig. 5. Dynamics of short synchronization

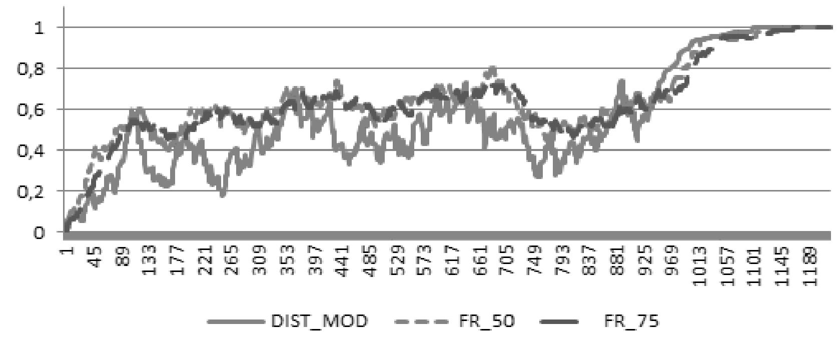

Fig. 6. Dynamics of long synchronization

As indicated in figures 5 and 6 in a long synchronization mode, there exists a threshold value close to 0.8 which, when exceeded in a some number of subsequent cycles, results in fast synchronization. In the case of networks that synchronizes quickly, calculated frequencies exceed this threshold also shortly before full synchronization. The experimental analysis of different threshold values ranging within 0.6 to 0.85 for all

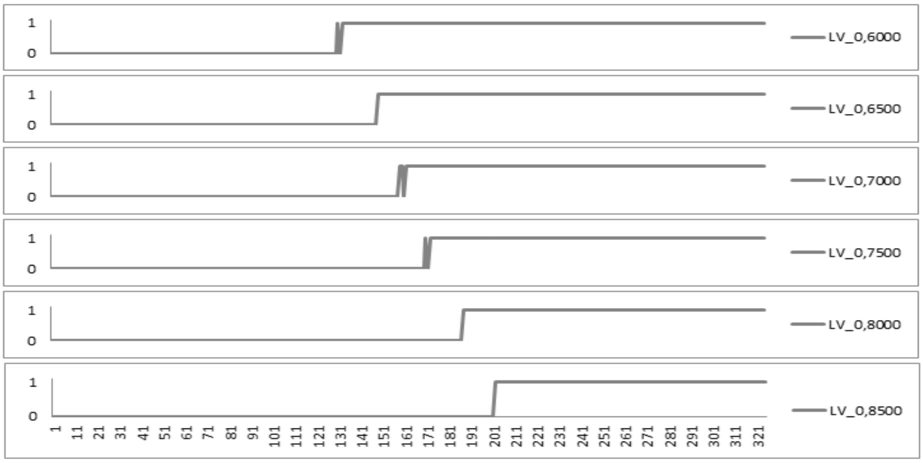

Fig. 7. Crossing the threshold for TPM with synchronization time 200

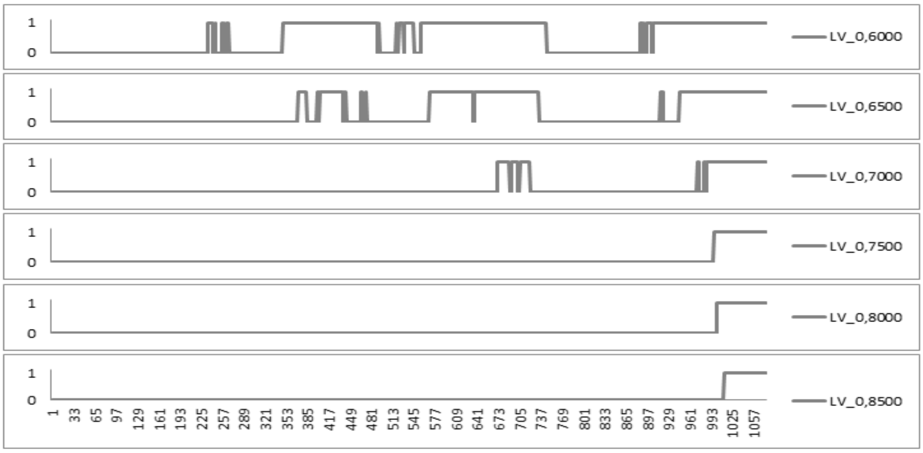

Fig. 8. Crossing the threshold for TPM with synchronization time over 900

analyzed frequencies is also conducted. The fact of passing over a given threshold can be presented in the following schemes for frequencies calculated in 75 previous steps (figures 7 and 8):

It is visible that frequencies generally increase during the network learning. This is consistent with an increase of compatibility of weights' vectors of the synchronizing networks. Thus the bigger threshold values are crossed by calculated frequencies after longer networks learning.

In the long synchronization only threshold 0.6 is exceeded before average learning time, which is 346 cycles for this TPM. Thus the use of the threshold set to 0.65 in this case indicates that there will be a high-speed synchronization. If this threshold is not exceeded in average or in third quartile time, it will be better to start synchronization with new, random weights.

4 Conclusions

The method of frequency analysis for exchange of equal results in TPM networks can be used to assess the synchronization level of both networks. As experimentally shown in this paper the calculated frequencies are not only strongly correlated with the distance between the weights' vectors but also are close to the values of modified distance. The latter is obtained by the small mean square error analysis. The charts containing the results of experiments presented herein shows that selecting the proper range for counting frequency and threshold (to be permanently exceeded) permits to specify whether one deals with either a short or a long synchronization.

References

1. Menezes, A., Vanstone, S., Van Oorschot, P.: Handbook of Applied Cryptography. CRC Press (1996)
2. Stokłosa, J., Bilski, T., Pankowski, T.: Data Security in Informatical Systems, PWN (2001) (in Polish)
3. Stinson, D.R.: Cryptography, Theory and Practice. CRC Press (1995)
4. Kanter, I., Kinzel, W., Kanter, E.: Secure Exchange of Information by Synchronization of Neural Networks. Europhys. Lett. 57, 141–147 (2002)
5. Klein, E., Mislovaty, R., Kanter, I., Ruttor, A., Kinzel, W.: Synchronization of Neural Networks by Mutual Learning and its Application to Cryptography. In: Advances in Neural Information Processing Systems, vol. 17, pp. 689–696. MIT Press, Cambridge (2005)
6. Klimov, A., Mityagin, A., Shamir, A.: Analysis of Neural Cryptography. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 288–298. Springer, Heidelberg (2002)
7. Ruttor, A.: Neural Synchronization and Cryptography, Ph.D. thesis, Würzburg (2006)
8. Ruttor, A., Kinzel, W., Naeh, R., Kanter, I.: Genetic Attack on Neural Cryptography. Phys. Rev. E 73(3), 036121–036129 (2006)
9. Dourlens, S.: Neuro-Cryptography. MSc Thesis, Dept. of Microcomputers and Microelectronics, University of Paris, France (1995)
10. Kotlarz, P., Kotulski, Z.: On Application of Neural Networks for S-Boxes Design. In: Szczepaniak, P.S., Kacprzyk, J., Niewiadomski, A. (eds.) AWIC 2005. LNCS (LNAI), vol. 3528, pp. 243–248. Springer, Heidelberg (2005)
11. Kanter, I., Kinzel, W.: Neural Cryptography, cond-mat/0208453 (2002)
12. Hassoun, M.: Fundamentals of Artificial Neural Networks. MIT Press (1995)
13. Osowski, S.: Neural Networks in Algorithmic Approach, WNT (1996) (in Polish)
14. Rutkowski, L.: Methods and Technics of Artificial Intelligence, PWN, Warsaw (2006) (in Polish)
15. Kanter, I., Kinzel, W.: The Theory of Neural Networks and Cryptography. In: Proceeding of the XXII Solvay Conference on Physics, The Physics of Communication, pp. 631–644 (2003)
16. Rosen-Zvi, M., Klein, E., Kanter, I., Kinzel, W.: Mutual Learning in a Tree Parity Machine and its Application to Cryptography. Phys. Rev. E 66, 066135 (2002)
17. Huntsberger, D.V.: Elements of Statistical Inference. Allyn and Bacon (1961)
18. Dolecki, M.: Tree Parity Machine Synchronization Time – Statistical Analysis. Mathematics, Physics and Informatics Series, Minsk 6(153), 149–151 (2012)
19. Dolecki, M., Kozera, R., Lenik, K.: The Evaluation of the TPM Synchronization on the Basis of their Outputs. Journal of Achievements in Materials and Manufacturing Engineering 57(2), 91–98 (2013)